Analyzing Repast Symphony models in R with **RRepast** package

Antonio Prestes García, Alfonso Rodríguez-Patón Aradas

Departamento de Inteligencia Artificial,
Universidad Politécnica de Madrid,
Campus de Montegancedo s/n,
Boadilla del Monte, 28660 Madrid, Spain

***Abstract*—In order to produce dependable results, the output of models must be carefully evaluated and compared to the experimental data. One of the main goals of analyzing a model is the understanding the effect of input factors on the model output. This task is carried out using a methodology known as sensitivity analysis. The analysis of Individual-based Models is hindered by the lack of simple tools allowing a complete and throughout evaluation without much effort. This kind of model tends to have a high level of complexity and the manual execution of a large experimental setup is generally not a feasible choice. Thus, it is required that model evaluation should ideally be simple and robust without demanding a high level of knowledge from modelers. In this work we present the RRepast, an open source GNU R package for executing, calibrating and analyzing Repast Symphony models directly from the R environment.**

I. Introduction

The individual-based modeling is being established progressively as a main-stream and valuable tool for modeling complex processes in many distinct areas of knowledge, ranging from social science, economics to any flavor of computational and systems science such as biology, ecology and so on [1]. The reason is, amongst other things, the relative ease with which detailed structural information can be incorporated into a model without the constraints of other methodologies [2]. Nonetheless, the possibility of incorporating many details comes with the cost of models with a high complexity levels, containing many rules and parameters for which the exact values are, in many cases, hard or impossible to determine experimentally, that is what is know as parameter uncertainty.

Model calibration is the task of estimate the set of values for input parameters of some simulation model which provides the best fitting to any empirical data set available for the system under study[3]. The estimation of acceptable values for the parameters of Individual-based Models and the analysis of uncertainty, requires specialized techniques which are complex computationally demanding. One of the objectives of these methods are understand the relative impact of input parameters on the overall model outcomes. According to [4] most of Individual-based models published tends to omit the systematic calibration and sensitivity analysis tasks, chiefly due the fact that modelers practitioners do not have the specific knowledge to implement or simply use the required methods. Therefore, it seems to be clear, that the availability of simple and user friendly tools for experiment design and analysis would help modelers to improve the formal quality of their models.

The Repast Symphony framework is a fast and flexible java-based environment with some built-in facilities for batching and parameter sweeping [5], widely used in many fields for building individual-based simulation models [6], [7], [8] of dynamic processes. Repast also has support for running GNU R [9], [10] code from inside the framework user interface but until now was not feasible running Repast models from R environment for controlling model in order to implement experimental designs, parameter calibration and sensitivity analysis, therefore hindering a throughout and comprehensive verification of Individual-based models.

In addition the real value of a computational model depends much on the ability of other researchers to reproduce and enhance the results elsewhere, in other words results must be reproducible. Hence, in order to achieve reproducibility, research methods should be stated clearly and should preferentially being backed by standard methods and software tools. In the following sections we will describe the **RRepast** package functionalities, the most significant API elements, as well as a worked example for illustrating the basic use case of the package.

II. The **RRepast** package

The **RRepast**[1] is an ongoing open source project developed primarily for invoking Repast Symphony models from inside GNU R environment, but having much more features added on top of this fundamental functionality, in order to make the analysis of Individual-based models developed with Repast, extremely straightforward, providing a powerful API which reduce the need to code the most commons methods. The package contains R and java code for linking the calls to the Repast subsystem. The software is delivered under the MIT license system.

The package has two main groups of functions: the first, directly related to the integration of Repast Symphony with R, allowing the instantiation, execution and control of a model

[1]The software can be found on the following CRAN URL: https://cran.r-project.org/web/packages/rrepast/

execution, as well as, gathering model output generated by any aggregated dataset defined into Repast model[11]. The second group of features relies on the first group for running model but exposing a complete set of methods for parameter calibration and for performing sensitivity analysis methods without much effort, including also functions for most common experimental design setups.

The first group of methods, are in turn subdivided into low and high level calls. The first type of them are the functions prefixed with the [Engine] keyword which wraps the calls to the java subsystem using the rJava package[12]. These functions are not intended for general use, instead the users should the high level calls which include, the calls depicted in the Table I.

The second group of functions inside the package contains *low level* functions for the design of experiments[13] by the user, as well as, *high level* methods which are the recommended entry point for the generation of experiments with the model. All of these *high level* functions have their names prefixed with the "Easy" keyword. The *Easy API* are designed to perform a complete and complex task with just one function call. Some of these functions are shown in the Table II and the current Easy API methods are presented in Table III.

III. **RRepast** IN ACTION

In this section we will provide some small examples on how to use the **RRepast** package for running Repast models and analyzing the data produced. In order to gets the model running from R code, some minimal steps must be carried out before calling Repast code.

1) Build an installer and install the Repast model.
2) Add the **rrepast-integration.jar** file, included in **RRepast** distribution, to the **lib** directory of the installed Repast model.
3) Add the integration configuration to scenario file in the .rs directory of the installed model. The integration consists in the following code: *<model.initializer class="org.haldane.rrepast.ModelInitializerBroker" />*

Once the previous steps are completed we are ready for running the model. The minimal code to execute the model is presented in Figure 1

```
library(rrepast)

# The directory where The repast model has been installed
install.dir<- "c:/models/themodel"

# Instantiate and load the model
obj<- Model(modeldir=install.dir, dataset="dataset", TRUE)

# Run the model
Run(obj)

# The model's output
output<- GetResults()
```

Figure 1: **The minimal code for running a Repast model from R. The boolean value in** *Model()* **tells RRepast to auto load the model's scenario.**

Function name	Description
Model(d, t, o, l)	This function creates an object instance for linking the Repast model to an R object. The required parameters are the directory where the model has been installed *(d)*, the duration of simulation in Repast ticks *(t)*, the name of any aggregated dataset of model for draining data generated by the model simulation*(o)* and a Boolean flag which tells the function to call the Load method. The default value is FALSE.
Load(m)	This function loads the Repast scenario from model's directory. The only required parameter *(m)* is an instance of Repast Model created with previous function.
Run(m, r, s)	The purpose of this function is to execute a single round of simulation using just one parameter set. The parameters for this function are a model instance *(m)*, the number of repetitions *(r)* and a collection the random seeds *(s)* to be used for each one of the repetitions. The only required parameter is the model instance, created with the **Model()** function. The default value for *r* is one.
RunExperiment(m, r, d, F)	Execute a complete experimental setup for different set of parameters. The parameters required are a model instance *(m)*, the number of replications *(r)*, the experimental design *(d)* and finally a user provided calibration function *(F)*. The experimental design parameter is an **R** data frame containing a complete set of model's parameter per row. The function returns a list with three data frame elements: the *paramset*, the *output* and *dataset* which holds respectively all simulated input parameters, the result of user provide calibration function and the complete dataset produced during the experiment execution.
GetSimulationParameters(e)	Returns the complete list of parameters declared by the model.
SetSimulationParameters(e, p)	Modify several parameters at once.
SaveSimulationData(t, e)	Exports the results of Run or RunExperiment to a csv or excel files.

Table I: The basic RRepast API Functions. These functions are used for loading, modifying the default parameters defined for model and for running the simulation.

In addition to the basic functionality for loading and running a model and retrieving the complete output of any dataset defined in the Repast model, the package contains an implementation of common techniques for screening and global sensitivity analysis as well as for verifying the stability of output variables. These functionalities are readily accessible, requiring very few lines of code. In the simplest case the modeler only has complete three tasks for getting the experiment done. The first one is to define a calibration function. The calibration function must return zero for the best fit and other number greater than zero otherwise. How the criteria are implemented is up to the modeler. That function is called internally by RRepast and has a specific format. The

Function name	Description
AddFactor(f, l, k, b, u)	Creates the parameter collection for the experimental setup. The function requires the data frame *(f)* where parameter will be added, if this parameter is not provided a new data frame will be created. The second parameter *(l)* is the random function used internally, the default value is *runif* which will be the valid choice in many cases, the next parameter is *k* the name of factor, the value provided must match some parameter defined in the repast model. The following two parameters *(b)* , *(u)* are the lower and the upper range, respectively. The function returns the updated *(f)* data frame with the new parameter.
AoE.RandomSampling(n, f)	Also known as Monte Carlo sampling, generate an experimental design based making random samplings of parameter space. The function takes two parameters, the sample size *(n)* and the factor *(f)* data frame created using **AddFactor()**. The function returns the design matrix form the provided parameters.
AoE.LatinHypercube(n, f)	Generates an experimental design using the Latin Hypercube stratified sampling technique which is more efficient sampling scheme, in terms of model evaluations, than the pure random sampling. The parameters *(n, f)* and return values are the same already described for the function **AoE.RandomSampling()**.
AoE.FullFactorial(n, f)	Creates a factorial design where the effects of all independent variables of model are studied simultaneously which implies many more model evaluations. The parameters *(n, f)* and return values are the same already described for the function **AoE.RandomSampling()**.
BuildParameterSet(d, p)	Constructs the data frame required for executing *RunExperiment()*. The function takes two parameters: the design matrix *(d)* created with one of previous functions and the declared parameters *(p)* defined in the Repast Model with the default values retrieve using the function **GetSimulation-Parameters()**. The functions returns a data frame with varying and fixed parameters for the experimental setup of choice.

Table II: The Experimental Setup API functions. These functions are used for experimental design, parameter calibration and sensitivity analysis.

Function name	Description
Easy.Stability(d,o,t,f,s,r,v,F)	Evaluate the behavior of model output in order to determine the minimum required number of replication of chosen experimental setup. The function accept the following parameters: the model installation directory *(d)*, the aggregated data source defined within the Repast model *(o)*, the simulation time in Repast ticks *(t)* which default value is 300 ticks, the input factors to be sampled *(f)* created with the previously mentioned function **AddFactor()**, the number of parameter samples *(s)*, the desired number of replications to be tried *(r)* being the default value 100, the output variables of interest which will be checked for their stability and convergence of the coefficient of variation *(v)*, this parameter is leaved empty all output variables are checked and finally the user provided calibration function *(F)* for determining the best input parameter combination.
Easy.Morris(d,o,t, f,p,s,r,F)	This function performs all required tasks for carrying out the method of Morris for screening. The parameters are practically the same as described for the previous function with exception of parameters *(p)* and *(s)* which are respectively the levels of input factors and the number of sampling points of Morris method[14].
Easy.Sobol(d,o,t,f,n,r,F)	Encapsulate all required steps for performing sensitivity analysis using Sobol method. The method of Sobol is a global sensitivity an analysis technique based on the decomposition of output variance [15], [14]. The parameter semantics are the same already described: the model installation directory *(d)*, the aggregated data source defined within the Repast model *(o)*, the simulation time in Repast ticks *(t)* , the input factors to be sampled *(f)*, the sample size *(n)*, the desired number of replications *(r)* and calibration function *(F)*.
Easy.Calibration(d,o,t,f,n,r,F)	This function estimate the best set of input parameters or factors performing a set of experiments in order to sample the calibration function. The objective of this function is to minimize the output of calibration function provided by the user.

Table III: The easy API functions. These functions are the preferred entry point for the eventual users. These "Easy" functions lump together a complete experiment task in just one call, reducing the coding needs to the minimum.

parameters passed to the function are the current set parameter used and the complete content of Model dataset output and the function must return a **cbind()** containing all individual criteria and optionally the sum of individual criteria.

In order to providing some more realist examples we have used the BactoSIM Repast model, which is an spatially explicit individual-based model for simulating the plasmid spread on a surface attached bacterial colony[16].

The **BactoSIM** simulation model has several parameters but we want to focus just on four of them keeping all other fixed. Thus, let's say, we want to evaluate the parameters named *gamma0*, *cyclePoint*, *conjugationCons* and *pilusExpression-Cost*. For accomplishing this task we will use the *Easy* API functions described in Table III. These functions return a list holding three elements:

- **experiment**. The experiment is also a list holding the parameter set (paramset), the calibration function output (output) and the experiment raw dataset (dataset). These three entities are connected by a column named **pset**.
- **object**. The reference to the object used which could be Morris or a Sobol instance.
- **charts**. Contains the reference to the plots generated.

Therefore, the first step could be to determine the required number of replications for the simulation experiments using

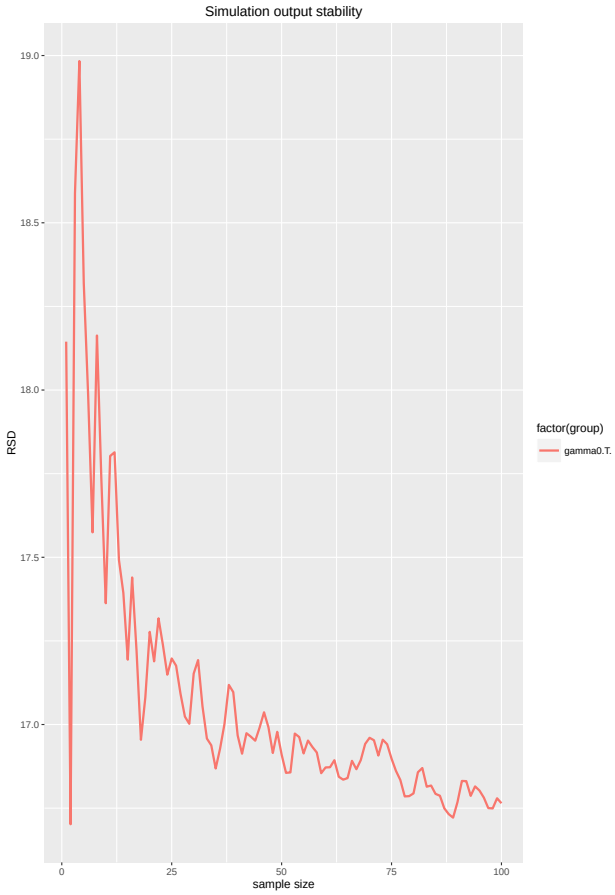

Figure 2: The stability of model output.It is possible to observe how, as far the number of replications of the experimental setup increases, the value of the coefficient of variation converges to a common value.

```
1
2  # The calibration function
3  fun<- function(p, r) {
4      criteria<- c()
5
6      Rate<- AoE.RMSD(r$Sim,r$Exp)
7
8      criteria<- cbind(Rate)
9      return(criteria)
10 }
11
12 # The factors under study
13 f<- AddFactor(name="cyclePoint",min=0,max=90)
14 ...
15 f<- AddFactor(f,name="gamma0",min=1,max=10)
16
17 v<- Easy.Morris("c:/BactoSim","out",300,f,50,10,10,fun)
```

Figure 3: The complete listing for perform the Morris's screening method. In the line 6 we define the **Rate** calibration criteria which is root-mean-square deviation between simulated and observed values. In lines 13 to 15 we create the input factor collection with their range of variation and finally line 17 shows the call of Easy.Morris function.

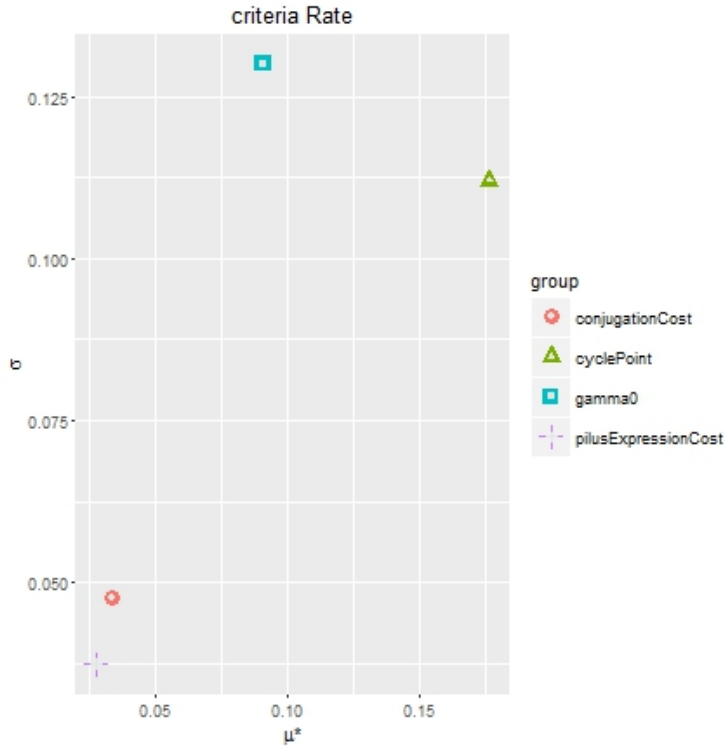

Figure 4: One of the output charts for Morris's screening method.The chart shows that the most import parameter for the Rate calibration metric is the cyclePoint followed by the gamma0.

the *Easy.Stability()* which output can be seen in Figure 2. The output shows on the abscissa the number of repetitions and on the ordinates the coefficient of variation for the desired output variable.

The listing shown in Figure 3 is an example of how easy is to analyze simulation experiments using RRepast. That is all code required to perform the Morris screening method for the BactoSIM Model. One of the outputs of Morris method is presented in Figure 4.

Finally we could decide, using the output of Morris method, to discard some of the parameters and focus only on those more important to perform the Sobol method. One of the output charts of Sobol method showing the indices and the confidence interval are show in Figure 5.

IV. Conclusions

In this report we have presented the basic aspects of **RRepast** package and how it could be used for perform the basic experimental setup of Repast Models. The API functions shown here are planned to be stable but they are not frozen yet as the project is still a work in progress, hence some slight variations may occur from version to version. Future versions will include more out-of-box functions for the statistical analysis of the model output and we are also evaluating the possibility of parallelize the multiple model's executions required by the sensitivity analysis methodologies.

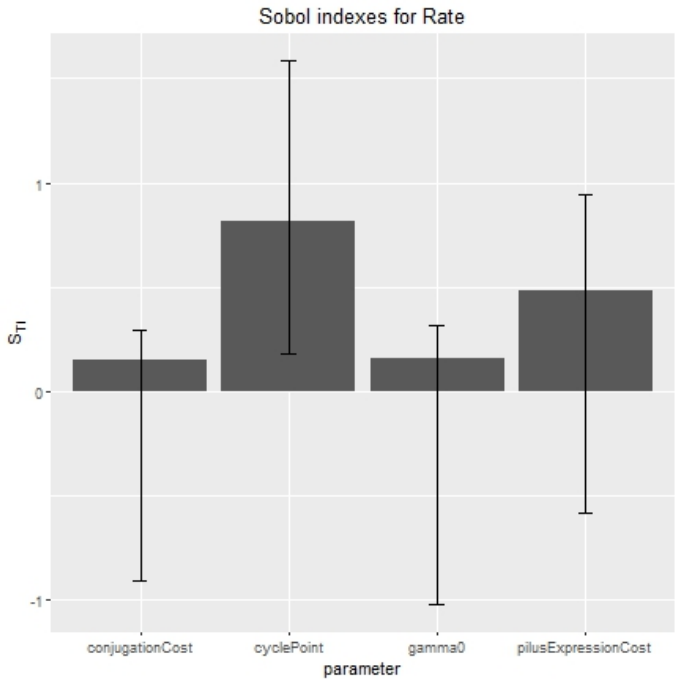

Figure 5: The output chart for Sobol method.The Sobol output shows that the dominant parameter is the cyclePoint but differently from Morris method the second in importance seems to be the pilusExpressionCost.

One of main drawback of analyzing individual-based models is the computational cost and the time required to complete an experimental setup for any model with a medium complexity level and a high number of agents being simulated. The simulations are safe and relatively easy to distribute as the same code will be executed for a different set of parameters but there are no need to communicate instances of experimental setup. Recently some interest has been shown on using Docker container technology for scientific research [17] and we are exploring that technology for easy deployment of the model execution across many nodes seamlessly.

Acknowledgments

This work was supported by the European FP7 - ICT - FET EU research project por: 612146 (PLASWIRES "Plasmids as Wires" project) www.plaswires.eu and by Spanish Government (MINECO) research grant TIN2012-36992.

References

[1] V. Grimm and S. F. Railsback, *Individual-based Modeling and Ecology: (Princeton Series in Theoretical and Computational Biology)*. Princeton: Princeton University Press, Jul. 2005. [Online]. Available: http://www.worldcat.org/isbn/069109666X

[2] F. L. Hellweger and V. Bucci, "A bunch of tiny individuals — Individual-based modeling for microbes," *Ecological Modelling*, vol. 220, no. 1, pp. 8–22, Jan. 2009. [Online]. Available: http://dx.doi.org/10.1016/j.ecolmodel.2008.09.004

[3] B. P. Zeigler, H. Praehofer, and T. G. Kim, *Theory of Modeling and Simulation, Second Edition*, 2nd ed. Academic Press, Jan. 2000. [Online]. Available: http://www.worldcat.org/isbn/0127784551

[4] J. C. Thiele, W. Kurth, and V. Grimm, "Facilitating Parameter Estimation and Sensitivity Analysis of Agent-Based Models: A Cookbook Using NetLogo and 'R'," *Journal of Artificial Societies and Social Simulation*, vol. 17, no. 3, 2014. [Online]. Available: http://jasss.soc.surrey.ac.uk/17/3/11.html

[5] M. North, N. Collier, J. Ozik, E. Tatara, C. Macal, M. Bragen, and P. Sydelko, "Complex adaptive systems modeling with Repast Simphony," *Complex Adaptive Systems Modeling*, vol. 1, no. 1, pp. 1–26, 2013. [Online]. Available: http://dx.doi.org/10.1186/2194-3206-1-3

[6] A. Watkins, J. Noble, R. Foster, B. Harmsen, and C. Doncaster, "A spatially explicit agent-based model of the interactions between jaguar populations and their habitats," *Ecological Modelling*, vol. 306, pp. 268–277, 2015.

[7] A. Gutfraind, B. Boodram, N. Prachand, A. Hailegiorgis, H. Dahari, and M. E. Major, "Agent-Based Model Forecasts Aging of the Population of People Who Inject Drugs in Metropolitan Chicago and Changing Prevalence of Hepatitis C Infections," *PLOS ONE*, vol. 10, no. 9, p. e0137993, sep 2015. [Online]. Available: http://dx.plos.org/10.1371/journal.pone.0137993

[8] I. L. Tack, F. Logist, E. Noriega Fernández, and J. F. Van Impe, "An individual-based modeling approach to simulate the effects of cellular nutrient competition on Escherichia coli K-12 MG1655 colony behavior and interactions in aerobic structured food systems," *Food Microbiology*, vol. 45, pp. 179–188, 2015.

[9] R Core Team, *R: A Language and Environment for Statistical Computing*, R Foundation for Statistical Computing, Vienna, Austria, 2015. [Online]. Available: https://www.R-project.org/

[10] M. J. Crawley, *The R Book*, 1st ed. Wiley, Jun. 2007. [Online]. Available: http://www.worldcat.org/isbn/0470510242

[11] M. J. North, N. T. Collier, J. Ozik, E. R. Tatara, C. M. Macal, M. Bragen, and P. Sydelko, "Complex adaptive systems modeling with Repast Simphony," *Complex Adaptive Systems Modeling*, vol. 1, no. 1, p. 3, 2013. [Online]. Available: http://www.casmodeling.com/content/1/1/3

[12] S. Urbanek, *rJava: Low-Level R to Java Interface*, 2016, r package version 0.9-8. [Online]. Available: https://CRAN.R-project.org/package=rJava

[13] C. R. Hicks, *Fundamental Concepts in the Design of Experiments*, 4th ed. Oxford University Press, USA, Mar. 1993. [Online]. Available: http://www.worldcat.org/isbn/0195122739

[14] G. Pujol, B. Iooss, A. J. with contributions from Sebastien Da Veiga, J. Fruth, L. Gilquin, J. Guillaume, L. L. Gratiet, P. Lemaitre, B. Ramos, and T. Touati, *sensitivity: Sensitivity Analysis*, 2015, r package version 1.11.1. [Online]. Available: https://CRAN.R-project.org/package=sensitivity

[15] A. Saltelli, S. Tarantola, F. Campolongo, and M. Ratto, *Sensitivity Analysis in Practice: A Guide to Assessing Scientific Models*, 1st ed. Wiley, Apr. 2004. [Online]. Available: http://www.worldcat.org/isbn/0470870931

[16] A. P. García and A. Rodríguez-Patón, "BactoSim — An Individual-Based Simulation Environment for Bacterial Conjugation." Springer International Publishing, 2015, pp. 275–279. [Online]. Available: http://link.springer.com/10.1007/978-3-319-18944-4{_}26

[17] C. Boettiger, "An introduction to Docker for reproducible research," *ACM SIGOPS Operating Systems Review*, vol. 49, no. 1, pp. 71–79, jan 2015. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2723872.2723882